une formalisation des faisceaux et des schémas affines en théorie des types avec Coq

Laurent Chicli

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Une formalisation des faisceaux et des schémas affines en théorie des types avec Coq

Laurent Chicli

N° 4216

Juin 2001

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4216--FR

INRIA
SOPHIA ANTIPOLIS

Une formalisation des faisceaux et des schémas affines en théorie des types avec Coq

Laurent Chicli

Thème 2 — Génie logiciel
et calcul symbolique
Projet Lemme

Rapport de recherche n° 4216 — Juin 2001 — 41 pages

Résumé : On présente ici un début de formalisation de la géométrie algébrique en théorie des types avec la définition des schémas affines. Pour cela topologie, algèbre et théorie des faisceaux ont étés développées en *Coq*, nous décrivons leurs spécifications, ainsi que les problèmes rencontrés lors de ce travail.

Mots-clés : Algèbre, Coq, Ctcoq, formalisation des mathématiques, faisceaux, schémas, topologie, théorie des types

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

A formalization of sheaves and affine schemes in Type Theory using Coq

Abstract: We present a formalization in Type theory of affine schemes, first step toward modern algebraic geometry. Topologie, algebra, and sheaves theories have been developed in *Coq*, we describe their specifications and the problems we have encountered.

Key-words: Algebra, Coq, Ctcoq, formalization, sheaves, schemes, topology, type theory

1 Introduction

Nous présentons nos spécifications et preuves en théorie des types des notions mathématiques nécessaires à la définition des schémas affines, l'un des premiers objets de la géométrie algébrique moderne.

Ce travail est basé sur la formalisation de l'algèbre effectuée par Loïc Pottier [POT99], qui comprends les notions de catégorie, d'ensemble, groupe, anneau, et corps, et est effectué avec la version 6.2.3 du logiciel d'aide à la démonstration *Coq* [COQ99] (et donc avec le *Calcul des Constructions Inductives* comme théorie des types).

Ce rapport s'articule autour de six parties. La première décrit la représentation des ensembles que nous utilisons, et présente certaines possibilités de *Coq* comme les coercions ou la synthèse automatique d'arguments implicites. On continue ensuite avec différentes notions de topologie comme celles d'intérieur, d'adhérence, de topologie engendrée ou de continuité. Puis avec l'algèbre: idéaux premiers, maximaux, le radical d'un idéal, les anneaux locaux et la localisation d'anneaux font partie des notions nécéssaires pour commencer à formaliser les schémas affines. La quatrième partie, la plus importante de ce travail, décrit nos spécifications de la théorie des faisceaux et se termine par la construction du faisceau associé à un préfaisceau. On s'y est fortement inspiré du livre 'Algebraic Geometry' [HAR77] de R. Hartshorne. Topologie, algèbre, et théorie des faisceaux sont ensuite réunis pour permettre la définition des schémas affines. Enfin on décrit dans la dernière partie les différents problèmes que nous avons rencontrés lors de ce travail.

L'interface graphique *CtCoq* [CRO96] pour *Coq* a été utilisée pour simplifier la création des preuves (avec 'proof-by-pointing' [BKT94]) et pour augmenter la lisibilité des énoncés (que nous voulons aussi proche que possible du langage mathématique). Dans la mesure où cela ne nuira pas à la compréhension de nos spécifications, c'est d'ailleurs l'affichage de *CtCoq* que nous utiliserons pour les présenter.

2 Quelques mots sur *Coq* et la représentation des ensembles

On présente ici quelques aspects de la formalisation de l'algèbre en *Coq* utilisée comme base de notre travail. Le lecteur interessé pourra se reporter à [POT99] pour plus de détails.

2.1 Ensembles et Sétoïdes

La manière de représenter les ensembles conditionne tout le travail et la formalisation future, il est donc essentiel que ce choix capture à la fois tous les ensembles mathématiques et qu'il soit suffisament flexible pour ne pas trop s'éloigner dans leur utilisation des mathématiques standards.

La notion de type semble a priori convenir pour représenter les ensembles : on peut voir un type comme l'ensemble des termes qui l'ont pour type, mais il n'existe malheureusement pas encore de notion de type quotient qui soit suffisament expressif pour englober les quotients mathématiques et leurs propriétés.

On utilise donc la structure de sétoïde (voir par ex. [SAI95]) pour représenter les ensembles. Un sétoïde est la donnée d'un type support, qui sera interprété comme le type des éléments de l'ensemble, et d'une relation d'équivalence, qui servira d'égalité sur cet ensemble. On peut ainsi appréhender de manière totalement homogène les ensembles quotients : l'ensemble E/R n'est rien d'autre que le sétoïde ayant le même type support que E mais où la relation initiale a été changée par R.

Definition.
a set is a structure with the following components:
Carrier: a type *(coercion)*
Equal: a type relation on Carrier
Prf_equiv: a proof that Equal is an equivalence *(coercion)*

L'image précédente correspond à ce qui apparait sous *CtCoq* avec un 'pretty-printer' approprié, elle correspond en fait au code *Coq* suivant :

```
Record Setoid: Type := {
Carrier:> Type;
Equal: (relation Carrier);
Prf_equiv:> (equivalence Equal) }.
```

Un mot avec cet exemple sur la syntaxe de *Coq*. La macro Record crée ici le type inductif `Setoid` ayant pour unique constructeur la fonction `Build_Setoid` qui demande trois arguments des types spécifiés dans les différents champs. Elle définit aussi les trois projections `Carrier, Equal, Prf_equiv` qui associeront à un sétoïde respectivement son type, sa relation, et la preuve qu'il s'agit bien d'une relation d'équivalence.

Le symbole `>` présent entre `Carrier` et `Type` permet de déclarer la projection `Carrier` comme coercion entre le type `Setoid` et le type `Type`: de cette manière un setoïde pourra être vu, à travers `Carrier`, comme un type. Plus précisément, si E est un setoïde, on pourra alors considérer un élément x ¨de type¨ E (alors que E n'est pas un type), parce que *Coq* l'instanciera automatiquement par `x:(Carrier E)`.

2.2 Sous-typage et parties d'un ensemble

Les parties comme prédicat

Le moyen le plus naturel de représenter une partie A d'un ensemble E est de se donner un prédicat P sur E de type `E->Prop` (A sera l'ensemble des éléments de E vérifiant P). Mais puisqu'on travaille avec les sétoïdes, il faut en plus demander aux prédicats que l'on considère d'être compatibles avec l'égalité de E. Soit donc la fonction `pred_compatible` qui associe à un prédicat P (de type `E->Prop`) la proposition $\forall x,y \in E$, *si* $(P\ x)$ *et* $(y = x)$ *alors* $(P\ y)$:

```
Definition pred_compatible: (E->Prop) ->Prop :=
   [P:E ->Prop] (x, y:E) (P x) -> (Equal y x) ->(P y).
```

On peut alors définir ce qu'est un prédicat dans le cadre des sétoïdes :

```
Record Predicate:= { Pred_fun: E ->Prop;
                     Pred_compatible_prf: (pred_compatible Pred_fun)}.
```

Les parties comme ensemble

Mais pour faire de la partie de E définie par un terme P de type `Predicate` un sétoïde, il faut pouvoir donner un type à ses éléments. Comme il n'existe pas en *Coq* de notion primitive de sous-typage, on procède ainsi :

Etant donné un type F et une fonction i de type `F->E`, on définit le setoïde `(subtype_set E F i)` par le type F et la relation d'équivalence xRy ssi $i(x) = i(y)$. Ce sétoïde est en bijection avec I, l'image de i, et peut donc être vu comme une partie de E.

Pour implémenter la partie de E définie par le prédicat P, il suffit alors de prendre pour F le type `subtype` suivant :

```
Record subtype :={subtype_elt : E
                  subtype_prf : (Pred_fun P subtype_elt)}
```

(les termes qui l'habitent sont les couples constitué d'un terme de type E et d'une preuve qu'il vérifie le prédicat P), et pour i la projection `subtype_elt : subtype->E`. Le terme `(subtype_set E subtype subtype_elt)` sera ainsi le sétoïde définit par le prédicat P.

On met en place les coercions nécessaires pour pouvoir voir un prédicat comme une partie et inversement. L'appartenance à une partie peut alors se définir comme la fonction `in_part` qui à un sétoïde E, un terme x de type E, et à un prédicat A sur E associe la proposition `(Pred_fun A x)`:

```
Definition in_part := [E:Setoid] [x:E] [A:(Predicate E)](Pred_fun A x).
```

L'ensemble des parties

Le sétoïde définissant l'ensemble des parties de E a pour type le type `Predicate` et pour égalité la double inclusion :

```
Definition eq_part: (Predicate E) -> (Predicate E) ->Prop :=
   [A:(Predicate E)] [B:(Predicate E)] (x:E)
   ((in_part x A) ->(in_part x B)) /\ ((in_part x B) ->(in_part x A)).

Lemma eq_part_equiv: (equivalence eq_part).
```

Et on peut alors définir l'ensemble des parties de E par :

```
Definition part_set: Setoid := (Build_Setoid eq_part_equiv).
```

Dans la définition de `part_set` ci-dessus on ne donne à *Coq* qu'un seul des trois arguments nécessaires à la fonction `Build_Setoid`. En effet, les deux arguments manquant (le type `(Predicate E)` et la relation `eq_part`) sont implicitement contenu dans le type de `eq_part_equiv`, et sont automatiquement synthétisés par *Coq*.

Remarquons que la fonction `subtype_elt` nous permet de voir un élément d'une partie comme un élément de l'ensemble au dessus : si `A:(part_set E)` et si `x:A` alors `(subtype_elt x):E`. Malheureusement *Coq* ne permet pas de déclarer cette fonction comme coercion (elle serait paramétrée par le setoïde E, ce qui n'est pas accepté par *Coq*).

On choisit d'afficher sous *CtCoq* $x \in A$ pour le terme `(in_part x A)`, $\mathcal{P}(E)$ pour `(part_set E)`, et de cacher la fonction `subtype_elt` pour plus de lisibilité.

3 Topologie

3.1 Topologie, ensembles ouverts comme type

Une topologie sur un ensemble E est une partie de l'ensemble des parties de E, contenant E et l'ensemble vide, et vérifiant deux propriétés : stabilité par union quelconque, et par intersection finie. On commence donc par définir ce qu'est pour un ensemble de parties la stabilité par union quelconque (la propriété `union_in_top` ci-dessous) et par intersection finie (`inter_in_top`).

Let E be a set.
Definition.
union_part: $\wp(\wp(E)) \Rightarrow \wp(\wp(E))$
1: `Intros P.` 1: `Apply` (!Build_Predicate ? x:E , $\exists$ A: $\wp$(E) , (A $\in$ P) $\wedge$ (x $\in$ A)).
Definition (union_in_top).
given top of type $\wp(\wp(E))$,
we define the proposition union_in_top(top) by
$\forall$ open_family: $\wp(\wp(E))$, open_family $\subset$ top $\Rightarrow$ union_part(open_family) $\in$ top

Definition (inter_in_top).
given top of type $\wp(\wp(E))$,
we define the proposition inter_in_top(top) by
$\forall$ open1,open2: $\wp(E)$, open1 $\in$ top $\Rightarrow$ open2 $\in$ top $\Rightarrow$ open1 $\cap$ open2 $\in$ top

Une topologie sur E est alors :

Definition.
a Topology is a structure with the following components:
top: of type Predicate($\wp$(E)) *(coercion)*
union_in_top_prf: a proof of union_in_top(top)
inter_in_top_prf: a proof of inter_in_top(top)
total_prf: a proof of full(E) $\in$ top
empty_prf: a proof of $\emptyset \in$ top

et un espace topologique est un ensemble donné avec une topologie:

Definition.
a topological space is a structure with the following components:
et_setoid: a set *(coercion)*
open: of type Topology(et_setoid)

On déclare une coercion entre le type `et` et le type `setoid`. Ainsi si `X:et` est un espace topologique alors un terme `x:X` (automatiquement coercé vers `x:(Carrier (et_setoid X))`) sera un élément du setoïde sous-jacent et un terme `U:(open X)` un élément de la topologie, donc un ouvert, comme en langage mathématique courant.

3.2 Adhérence, intérieur, bord

Les notions de fermés, adhérences, intérieurs, bords, ainsi que tout les lemmes associés s'expriment clairement. Par exemple, pour définir l'adhérence d'une partie A de E, on com-

mence par construire l'ensemble `closed_containing` des fermés contenant A, puis on prends l'intersection de ces fermés :

```
Let E be a topological space.
Let A be of type ℘(E).
Definition.
    closed_containing:℘(℘(E))
 1: Apply (!Build_Predicate ? F:℘(E), A ⊂ F ∧ F is closed).

Definition.
    let adh be intersection_part(closed_containing)
     (its type is ℘(E)).
```

On montre alors les lemmes classiques ainsi que la propriété caractéristique de l'adhérence : c'est le plus petit fermé qui contienne A.

```
Lemma (included_adh).
    A ⊂ adh.

Lemma (adh_closed).
    ∀ A:℘(E), adh(A) is closed.

Lemma (closed_containing_adh).
    ∀ F,G:℘(E), F is closed ⇒ G ⊂ F ⇒ adh(G) ⊂ F.
```

L'intérieur d'une partie A s'exprime de façon similaire : c'est l'union de tous les ouverts inclus dans A.

```
Definition.
    open_included:℘(℘(E))
 1: Apply (!Build_Predicate ℘(E) U:℘(E), U ⊂ A ∧ U is open).
Definition.
    let int be union_part(open_included)
     (its type is ℘(E)).
```

Et, comme pour l'adhérence :

Lemma (int_included).
$\forall$ A,B:$\wp$(E) , A $\subset$ B $\Rightarrow$ int(A) $\subset$ int(B).

Lemma (int_open).
$\forall$ A:$\wp$(E) , int(A) is open.

Lemma (eq_set_int).
$\forall$ A:$\wp$(E) , A is open $\Rightarrow$ A = int(A).

Lemma (open_included_int).
$\forall$ A,B:$\wp$(E) , B is open $\Rightarrow$ B $\subset$ A $\Rightarrow$ B $\subset$ int(A).

On peut alors définir le bord d'une partie et la propriété de densité :

Definition.
let border be the function defined by , for all A:Predicate(E) , adh(A) $\cap$ $\neg$int(A)
(its type is $\wp$(E) $\Rightarrow$ $\wp$(E)).
Definition.
let is_dense be the function defined by , for all A,B:$\wp$(E) , B $\subset$ adh(A)
(its type is $\wp$(E) $\Rightarrow$ $\wp$(E) $\Rightarrow$ Prop).

et montrer les lemmes généraux concernant ces notions.

3.3 Outils pour construire une topologie

On construit parfois les espaces topologiques que l'on considère en spécifiant non pas l'ensemble des ouverts mais celui des fermés, ou en considérant la topologie engendrée par une base d'ouvert. On décrit dans cette section les librairies qui nous permettent de telles constructions.

3.3.1 Topologie définie par l'ensemble des fermés

Si E est un sétoïde et F un ensemble de parties de E candidat à former l'ensemble des fermés d'une topologie sur E, il faut qu'il vérifie les quatres propriétés suivantes : le vide et E appartiennent à F, stabilité de F par intersection quelconque et union finie. On se donne donc en paramètre les preuves `F_empty`, `F_full`, `F_prop1`, et `F_prop2` de ces propositions :

Definition (infinite_inter_closed).
given close of type $\wp(\wp(E))$,
we define the proposition infinite_inter_closed(close) by
$\forall$ closed_family:$\wp(\wp(E))$,
closed_family $\subset$ close $\Rightarrow$ intersection_part(closed_family) $\in$ close
Definition (finite_union_closed).
given close of type $\wp(\wp(E))$,
we define the proposition finite_union_closed(close) by
$\forall$ closed1,closed2:$\wp(E)$,
closed1 $\in$ close $\Rightarrow$ closed2 $\in$ close $\Rightarrow$ (closed1 $\cup$ closed2) $\in$ close
Let F_empty be of type $\emptyset \in$ F.
Let F_full be of type full(E) $\in$ F.
Let F_prop1 be of type infinite_inter_closed(F).
Let F_prop2 be of type finite_union_closed(F).

Soit `complementary` la fonction qui à un ensemble de parties associe l'ensemble des complémentaires de ces parties :

Definition.
complementary:$\wp(\wp(E)) \Rightarrow \wp(\wp(E))$
1: `Intros` F. 1: `Apply` (!Build_Predicate ? C:$\wp(E)$, $\neg$C $\in$ F).

On peut alors démontrer les quatres lemmes suivant :

Lemma (topology_by_closed_1).
union_in_top(complementary(F)).

Lemma (topology_by_closed_2).
inter_in_top(complementary(F)).

Lemma (topology_by_closed_3).
full(E) $\in$ complementary(F).

Lemma (topology_by_closed_4).
$\emptyset \in$ complementary(F).

qui font de `(complementary F)` une topologie sur E:

Definition.
let Build_topology_by_closed be
(!Build_Topology
E complementary(F) topology_by_closed_1 topology_by_closed_2
topology_by_closed_3 topology_by_closed_4).

3.3.2 Topologie engendrée par une base d'ouverts

Dans notre developpement, une base d'ouverts B est, comme une topologie, une partie de l'ensemble des parties de E, contenant E et l'ensemble vide, mais ne vérifiant que la stabilité par rapport aux intersections finies. C'est justement en considérant l'ensemble des parties engendrées par union quelconque d'éléments de B que l'on va obtenir une topologie dite 'engendrée par B'.

Definition.
a open_base is a structure with the following components:
open_base_setoid: of type Predicate($\wp$(E)) *(coercion)*
base_inter_in_top_prf: a proof of inter_in_top(open_base_setoid)
base_total_prf: a proof of full(E) $\in$ open_base_setoid
base_empty_prf: a proof of $\varnothing \in$ open_base_setoid

Et on définit :

Let B be of type open_base.
Definition.
generated_part_by_union: $\wp(\wp(E))$
1: `Apply` (!Build_Predicate ? x:$\wp$(E), $\exists$ Y:$\wp(\wp(E))$, Y $\subset$ B $\wedge$ (x = union_part(Y))).

Il ne reste plus qu'à démontrer que `generated_part_by_union` est bien une topologie :

Lemma (full_in_top_for_bases).
full(E) $\in$ generated_part_by_union.

Lemma (empty_set_in_top_for_bases).
$\varnothing \in$ generated_part_by_union.

Lemma (union_in_top_for_bases).
union_in_top(generated_part_by_union).

Lemma (inter_in_top_for_bases).
inter_in_top(generated_part_by_union).

pour pouvoir définir la topologie engendrée :

Definition.
let generated_topology_o be
(!Build_Topology
E generated_part_by_union(B) union_in_top_for_bases inter_in_top_for_bases
full_in_top_for_bases emptyset_in_top_for_bases)
(its type is Topology(E)).

3.4 Continuité

On garde la définition la plus générale de la continuité : une fonction $f : E \to F$ est continue en un point a si quelque soit U ouvert de F contenant $f(a)$, $f^{-1}(U)$ est un ouvert de E.

Definition (cont_in_pt).
given f of type (Map E F) and a of type E ,
we define the proposition (cont_in_pt f a) by
$\forall$ W:open of F , f(a) $\in$ W $\Rightarrow$ f^{-1}(W) is open

Et une application est continue si elle est continue en chacun des points de son domaine de définition:

Definition (cont).
given f of type (Map E F) ,
we define the proposition cont(f) by $\forall$ W:open of F , f^{-1}(W) is open
Lemma (cont_then_conteverywhere).
$\forall$ f:(Map E F) , f is continuous $\leftrightarrow$ $\forall$ a:E , (cont_in_pt f a).

On fait bien sûr le lien avec les autres notions, équivalentes, de continuité (image inverse d'un fermé, caractérisation par l'adhérence):

Lemma (cont_with_closed).
$\forall$ f:(Map E F) , f is continuous $\Rightarrow$ $\forall$ C:closed(F) , f^{-1}(C) is closed.

Lemma (im_of_adh).
$\forall$ f:(Map E F) , f is continuous $\Rightarrow$ $\forall$ C:$\wp$(E) , f(adh(C)) $\subset$ adh(f(C)).

Lemma (cont_with_adh_inv).
$\forall$ f:(Map E F) , ($\forall$ C:$\wp$(F) , f^{-1}(adh(C)) = adh(f^{-1}(C))) $\Rightarrow$ f is continuous.

4 Algèbre

Comme son nom l'indique, la géométrie algébrique à pour objet de discuter 'algébriquement' de questions géométriques. Il est donc nécessaire de dévelloper des librairies d'algèbre aussi complètes que possible. En particulier les anneaux, les idéaux, et certaines de leur propriétés vont jouer un rôle primordial. On décrit dans ce chapitre les notions d'idéaux premiers, maximaux, les radicaux par rapport à une partie multiplicative quelconque, et notre formalisation de la localisation d'anneaux et d'anneaux locaux.

4.1 Idéaux, idéaux premiers, maximaux

Idéaux premiers. Un idéal premier I est un idéal propre vérifiant la propriété suivante : si $x * y$ appartient à I alors x ou y appartient à I.

Let R be a commutative ring.
Definition.
let is_prime be
the function defined by , for all I:ideal(R) , $\forall$ x,y:R , x*y $\in$ I $\Rightarrow$ (x $\in$ I) $\vee$ (y $\in$ I).
Definition.
let propre be the function defined by , for all I:$\wp$(R) , I $\neq$ full(R).

Definition.
a prime_ideal is a structure with the following components:
prime_ideal_ideal: of type ideal(R) *(coercion)*
prime_ideal_prop: a proof of prime_ideal_ideal is a prime ideal
prime_ideal_propre: a proof of prime_ideal_ideal is a proper ideal

Lemma (prime_ideal_propre2).
$\forall$ p:prime_ideal , $1_R \notin$ p.

Lemma (is_prime_reverse).
$\forall$ p:prime_ideal , $\forall$ x,y:R , y $\notin$ p $\wedge$ x $\notin$ p $\Rightarrow$ x*y $\notin$ p.

D'autres lemmes :

Lemma (inter_included_prime).
$\forall$ p:prime ideal of R , $\forall$ I,J:ideal(R) , (I $\cap$ J) $\subset$ p $\Rightarrow$ (I $\subset$ p) $\vee$ (J $\subset$ p).

Puis les lemmes qui caractérisent les idéaux premiers en termes d'anneaux quotients (attention : ici la proposition `I is a prime ideal` correspond simplement à la proposition `(is_prime I)`).

Definition.
let idomain_proposition be
the function defined by , for all R:CRING , ∀ x,y:R , x ≠ 0R ⇒ y ≠ 0R ⇒ x*y ≠ 0R.

Lemma (prime_integral).
I is a prime ideal ⇒ R/I is an integral domain.

Lemma (integral_prime).
R/I is an integral domain ⇒ I is a prime ideal.

Idéaux maximaux. Un idéal maximal est un idéal propre et maximal pour la relation d'inclusion entre idéaux :

Definition.
let is_maximal be
the function defined by , for all I:ideal(R) , ∀ J:ideal(R) , I ⊂ J ⇒ I=J ∨ (J = full(R)).

Definition.
a maximal_ideal is a structure with the following components:
maximal_ideal_ideal: of type ideal(R) *(coercion)*
maximal_ideal_prop: of type maximal_ideal_ideal is a maximal ideal
maximal_ideal_propre: of type maximal_ideal_ideal is a proper ideal

Une autre façon d'exprimer qu'un idéal I est maximal est de dire que l'idéal engendré par I et un élément quelconque x de R n'appartenant pas à I est l'anneau R tout entier:

Lemma (maximal_prop1).
I is a maximal ideal ⇒ ∀ x:R , x ∉ I ⇒ full(R) = I+<{x}>.

Lemma (maximal_prop1_rev).
(∀ x:R , x ∉ I ⇒ full(R) = I+<{x}>) ⇒ I is a maximal ideal

On définit ensuite la proposition `field_prop` : R est un corps si tout élément non nul est inversible :

Definition.
invertible:Predicate(R)
1: `Apply` (!Build_Predicate ? x:R , ∃ z:R , (x*z = 1R) ∧ (z*x = 1R)).
Definition.
let field_prop be
the function defined by , for all R:ring , ∀ x:R , x ≠ 0R ⇒ x ∈ invertible(R).

Et à l'aide des lemmes précédents on peut démontrer la caractérisation des idéaux maximaux en termes d'anneaux quotients :

Lemma (maximal_field_prop).
I is a maximal ideal $\Rightarrow$ R/I is a field

4.2 Opérations sur les idéaux

Considérons un anneau commutatif R, on commence par construire le sétoïde `Set_Ideal` des idéaux de R (l'égalité y est évidemment celle des parties de R) :

Let R be a commutative ring.
Definition.
Set_Ideal:set
1: `Apply` (!Build_Setoid ideal(R) !Equal($\wp$(R))).

Somme d'idéaux

Soit `ideal_family` une famille d'idéaux dont on veut définir la somme :

Let ideal_family be of type $\wp$(Set_Ideal).

La somme des idéaux de `ideal_family` est l'intersection de tous les idéaux qui les contiennent tous. On construit donc d'abord l'ensemble `ideal_containing` des idéaux qui contiennent tous les idéaux de `ideal_family`, puis on en prends l'intersection. Notons qu'à ce niveau on ne peut pas utiliser la fonction `intersection_part` qui construit justement l'intersection de parties de R : une famille d'idéaux n'est pas vu comme un ensemble de parties de R.

Definition.
ideals_containing: $\wp$(Set_Ideal)
1: `Apply` (!Build_Predicate Set_Ideal a:ideal(R) , $\forall$ i:ideal_family , i $\subset$ a).
Definition.
sum_ideals:ideal(R)
1: `Vinyl2` x:R , $\forall$ a:ideals_containing , x $\in$ a.

On montre ensuite les deux propriétés caractéristiques de la somme d'idéaux. La première : tous les idéaux dont on a fait la somme sont inclus dans la somme :

Lemma (sum_ideals_prop1).
$\forall$ a:ideal_family , a $\subset$ sum_ideals.

La seconde : la somme des idéaux est bien le plus petit idéal contenant tous les idéaux dont on a fait la somme.

Lemma (sum_ideals_prop2).
∀ I:ideal(R) , (∀ J:ideal_family , J ⊂ I) ⇒ sum_ideals ⊂ I.

4.3 Radicaux

Élévation à la puissance dans un anneau

La fonction `ring_power` ci-dessous élève l'élément r à la puissance n (on pose comme classiquement en mathématique $r^0 = 1_R$) :

Let R be a ring.
Let r be of type R.
Definition:
Let ring_power be the recursive function, with value into R,
mapping n:nat to:
Cases of n :

O=>1R
(S n')=>ring_power(n')*r

Quelques lemmes :

Lemma (ring_power_comp).
∀ R:ring , ∀ n1,n2:nat , ∀ x,y:R , x = y ⇒ <nat> n1 = n2 ⇒ x ^ n1 = y ^ n2.

Lemma (ring_one_power).
∀ R:ring , ∀ n:nat , 1R ^ n = 1R.

Lemma (ring_zero_power).
∀ R:ring , ∀ n:nat , n >=1 ⇒ 0R ^ n = 0R.

Pour un anneau commutatif on a de plus :

Lemma (cring_power_morphism).
∀ R:cring , ∀ n:nat , ∀ x,y:R , x*y ^ n = x ^ n*y ^ n.

Enfin un idéal $\mathfrak{p}$ premier vérifie :

Lemma (prime_power).
∀ R:cring , ∀ x:R , ∀ p:prime ideal of R , ∀ n:nat , x ^ n ∈ p ⇒ x ∈ p.

Parties multiplicatives

Une partie multiplicative d'un anneau R est une partie contenant l'unité et stable pour la multiplication :

Let R be a ring.
Definition.
a multiplicative_part is a structure with the following components:
multi_part: of type Predicate(R) *(coercion)*
multi_part_one: of type 1R ∈ multi_part
multi_part_prop: of type ∀ x,y:R , x ∈ multi_part ⇒ y ∈ multi_part ⇒ x*y ∈ multi_part

Si M est une telle partie multiplicative, on fournit les fonctions qui permettent de voir respectivement le produit de deux éléments de M et l'unité comme terme de type M.

Let M be of type multiplicative_part.
Definition.
let build_elt_mp_prod be the function defined by , for all m1,m2:M , (m1*m2[in M]).
Definition.
let multi_part_unit be Build_subtype(multi_part_one(M)).

Et on construit `multi_part_min` la partie multiplicative minimale : celle qui ne contient que 1_R.

Definition.
multi_part_min:multiplicative_part
1: `Apply` !Build_multiplicative_part({1R}). 1: Abstract (`Auto with` algebra).

Radical

On peut maintenant définir le radical d'un idéal relativement à une partie multiplicative M quelconque :

Definition.
radical:$\wp$(R)
1: `Apply` (!Build_Predicate ? x:R , ∃ m:M , ∃ n:nat , m*x ^ n ∈ I).

On peut démontrer les lemmes suivants :

Lemma (included_radical).
I ⊂ radical.
1: `Red`.

Lemma (radical_increasing).
I ⊂ J ⇒ (radical I M) ⊂ (radical J M).

Et si R est commutatif :

Lemma (radical_multiplicative).
∀ R:cring ,
∀ M:multiplicative part of R ,
∀ I:ideal(R) , ∀ x:(radical I M) , ∀ y:R , x*y ∈ (radical I M).

Pour le radical relatif à la partie multiplicative minimale `(radical I (multi_part_min R))` et que l'on note $\sqrt{I}$ sous *Ctcoq*, on a aussi :

Lemma (radical_propre).
√I = full(R) ⇒ I = full(R).

Lemma (radical_to_natural_def).
∀ x:R , x ∈ √I ⇒ ∃ n:nat , x ^ n ∈ I.

Lemma (natural_def_to_radical).
∀ x:R , (∃ n:nat , x ^ n ∈ I) ⇒ x ∈ √I.

4.4 Localisation

4.5 Anneaux locaux

4.5.1 Définition et spécification

Un anneau local est un anneau qui ne contient qu'un idéal maximal, que l'on a souvent l'habitude en mathématique de considérer. Pour mimer ceci on peut définir une anneau local comme la donnée (toujours via la macro `Record`) d'un anneau R, d'un idéal maximal $\mathfrak{m}$ de R, et d'une preuve d'unicité des idéaux maximaux : si $\mathfrak{m}_1$ et $\mathfrak{m}_2$ sont deux idéaux maximaux

de R, alors $\mathfrak{m}_1 = \mathfrak{m}_2$.

Definition.
let is_local be
the function defined by , for all R:cring , ∀ m1,m2:maximal ideal of R , m1=m2.
Definition.
a local_ring is a structure with the following components:
lr_ring: a commutative ring *(coercion)*
lr_maximal_ideal: a maximal ideal of R
lr_prop: of type is_local(lr_ring)

Cependant plus tard dans notre developpement (lors de la définition des espaces annelés) on a besoin d'écrire un terme qui énonce que "tel anneau R est local", et si `(is_local R)` semble correspondre, cela ne suffit cependant pas pour voir l'anneau en question comme un anneau local (puisqu'il manque la donnée de l'idéal maximal). On doit donc rennoncer à inclure la donnée de l'idéal maximal dans la définition des anneaux locaux :

Definition.
a local_ring is a structure with the following components:
lr_ring: a commutative ring *(coercion)*
lr_prop: of type is_local(lr_ring)

En contrepartie, on ne pourra pas parler de l'idéal maximal d'un anneau local, et les lemmes et propriétés de l'unique idéal maximal d'un anneau local R devront en fait être quantifiés sur tous les idéaux maximaux de R. Par exemple, le théorème qui énonce que l'ensemble des inversibles d'un anneau local est le complémentaire de son idéal maximal est formalisé ainsi :

Lemma (local_ring_invertibles).
∀ R:local_ring , ∀ M:maximal ideal of R , invertible(R) = ¬M.

On se donne quand même la possibilité de choisir un idéal maximal dans un anneau pour démontrer des propositions, avec l'axiome suivant :

Axiom ex_maximal_ideal:
∀ R:cring , ∀ I:ideal(R) , I is a proper ideal ⇒ ∃ M:maximal ideal of R , I ⊂ M.

à "valeur" dans `Prop` (il permet d'énoncer l'existence de l'idéal, pas de le construire).

4.5.2 Le localisé en un idéal premier comme anneau local

On peut alors définir le localisé d'un anneau en un idéal premier $\mathfrak{p}$ comme anneau local : on commence par construire le complémentaire de $\mathfrak{p}$ comme partie multiplicative :

```
Let R be a commutative ring.
Let p be a prime ideal of R.
Definition.
    mp_compl_prime:multiplicative part of R
 1: Apply (!Build_multiplicative_part R ¬p).
```

L'idéal maximal du localisé est en fait l'ensemble des fractions $\frac{x_1}{x_2}$ tels que $x_1 \in \mathfrak{p}$:

```
Definition.
    localize_prime_maximal:maximal ideal of localize(mp_compl_prime)
 1: Vinyl x:localize(mp_compl_prime), x1 ∈ p.
```

On montre alors qu'il est le seul idéal maximal du localisé :

```
Lemma (local_prime_proof1).
    ∀ m1:maximal ideal of localize(mp_compl_prime), m1=localize_prime_maximal.
```

ce qui fait du localisé un anneau local :

```
Definition.
    localize_prime:local_ring
 1: Apply !Build_local_ring(localize(mp_compl_prime)).
```

5 Théorie des faisceaux

5.1 Catégorie associée à un espace topologique

5.1.1 Définition

Soit X un espace topologique. Pour définir la notion de préfaisceau sur X, on doit d'abord définir la catégorie Top_cat dont les objets sont les ouverts de X, et dont l'ensemble des morphismes $Hom(U,V)$ entre deux ouverts est soit le singleton contenant l'injection canonique $U \hookrightarrow V$ si $U \subset V$, soit l'ensemble vide.

5.1.2 Une première idée

Formaliser l'ensemble des objets ne pose pas de problèmes, il s'agit de la topologie elle-même. Mais pour l'ensemble des morphismes, la discution sur l'inclusion éventuelle de U

dans V mérite attention. La première idée est de se donner en paramètre une fonction de décision sur l'inclusion :

Parameter (incl_dec). ∀ U,V:open of X , { U ⊂ V }+{ ¬U ⊂ V }

Dans le *Calcul des Constructions Inductives* l'existence d'une telle fonction signifie que l'on peut décider constructivement (c'est-à-dire sans utiliser le tiers exclus ou l'axiome du choix) de l'inclusion de deux ouverts et permet de construire d'autres objets en raisonnant par cas. On peut donc s'en servir pour définir $Hom(U,V)$.

Cette spécification a deux désavantages. Le premier est que cette fonction de décision doit être dans la plupart des cas considérée comme un axiome car l'inclusion entre deux parties d'un ensemble n'est pas décidable en général. Le second est qu'elle oblige régulièrement à travailler avec des cas vides (dans le cas où il n'y a pas inclusion $Hom(U,V) = \emptyset$), cas résolubles aisément avec *Coq*, mais pénibles à la longue et que l'on ne considère jamais en mathématiques. Pour définir la composition de morphismes $Hom(U,V) \times Hom(V,W) \rightarrow Hom(U,W)$ on doit par exemple distinguer quatre cas selon les différentes possibilités d'inclusion de U dans V et V dans W.

5.1.3 Spécifications

Notre solution (due à André Hirschowitz) consiste à considérer $Hom(U,V)$ comme le setoïde dont les éléments sont les preuves, toutes égales, que $U \subset V$. Ceci est tout à fait possible dans le *Calcul des Constructions Inductives* où les preuves sont des objets comme les autres. Ainsi que U soit ou non inclus dans V la spécification de $Hom(U,V)$ correspondra bien à sa définition mathématique: dans le premier cas ce sera un singleton, dans l'autre l'ensemble vide.

Let U, V be open of X .
Definition.
a prf_included is a structure with the following components:
prf_included_prf: of type U ⊂ V *(coercion)*

Definition.
let eq_triv be the function defined by , for all t1,t2:prf_included , True
(its type is prf_included ⇒ prf_included ⇒ Prop).
Lemma (eq_triv_equiv).
eq_triv is an equivalence.

Definition.
let HomTop be the set Build_Setoid(eq_triv_equiv).

Cette fois-ci, aucun axiome n'est nécessaire, et seuls les cas significatifs mathématiquement, ceux où U est bien inclus dans V, doivent être étudiés. Par exemple, pour définir la composition de deux morphismes on doit construire un élément de `(HomTop U W)` à partir d'un élément de `(HomTop U V)` et d'un élément de `(HomTop V W)`, c'est à dire une preuve de $U \subset V$ sous les hypothèses $U \subset V$ et $V \subset W$, ce qui est immédiat.

5.2 Foncteurs contravariants et préfaisceaux

On a choisi de formaliser les préfaisceaux et faisceaux à valeur dans la catégorie CRING des anneaux commutatifs. On ne peut à priori pas généraliser dans *Coq* la notion de préfaisceaux à toute catégorie C parce qu'on y utilise l'égalité des setoïdes. Il suffit cependant de remplacer dans notre developpement le mot clé CRING par celui ne n'importe quelle autre catégorie dont les objets peuvent être coercé vers les setoïdes pour obtenir la notion de préfaisceau correspondante. La seule restriction concerne les spécifications des structures d'anneaux induites, qui devront évidemment être adaptés.

Un préfaisceau d'anneaux commutatifs sur un espace topologique X est un foncteur contravariant de la catégorie `(Top_cat X)` dans la catégorie CRING des anneaux commutatifs.

Les foncteurs contravariants entre deux catégories C_1 et C_2 sont définit comme dans [SAI95], en se donnant une application entre les objets et une autre entre les morphismes, satisfaisant deux propriétés: $F(id_{C_1}) = id_{C_2}$, et $F^*(goh) = F^*(h)oF^*(g)$.

Definition.
a functor is a structure with the following components:
fctr_ob: a function of type Ob(C1) ⇒ Ob(C2) *(coercion)*
fctr_morph: of type ∀ a,b:Ob(C1) , (a ⟶ b) ⇝ (fctr_ob(a) ⟶ fctr_ob(b))
im_of_id_prf: of type ∀ a:Ob(C1) , (fctr_morph a a)(Id$_a$) = Id$_{\text{fctr_ob}(a)}$
distrib_prf: of type ∀ a,b,c:C1 ,
∀ fa:a ⟶ b ,
∀ fb:b ⟶ c ,
(fctr_morph a c)(fb o fa) =
((fctr_morph b c)(fb) o (fctr_morph a b)(fa))

Et on peut finalement écrire:

Definition.
let CRpresheaf be (Cfunctor Top_cat(X) CRING).

Un préfaisceau F associe donc à chaque ouvert U de X un anneau $F(U)$, et à chaque inclusion $U \subset V$ un morphisme d'anneaux $F^*(U \subset V) : F(V) \to F(U)$, dit morphisme de restric-

tion, que l'on notera ρ_{VU}. Si $s \in F(V)$, on note aussi $s_{|U}$ l'élément $F^*(U \subset V)(s)$ de $F(U)$. Toutes ces notations sont possibles avec *CtCoq* , les énoncés restent ainsi lisibles, malgré la complexité grandissante du code. Par exemple, le terme qui apparait sous *CtCoq* comme :

```
s|W = s|V|W.
```

correspond en fait au code :

```
(Equal
     (((Cfctr_morph F W U) (Build_prf_included (included_trans pv pu))) s)
     (((Cfctr_morph F W V) pv) (((Cfctr_morph F V U) pu) s))).
```

5.3 Faisceaux

5.3.1 Définition

Un faisceau est un préfaisceau vérifiant les deux propriétés suivantes:

- *propriété d'unicité*: si s,$s' \in F(U)$ et si $\{V_i\}_{i\in I}$ est un recouvrement ouvert de U tel que $\forall i\ s_{|V_i} = s'_{|V_i}$ alors $s = s'$ dans $F(U)$,
- *propriété de recollement*: si on a une collection d'éléments $s_i \in F(V_i)$, $\{V_i\}_{i\in I}$ étant toujours un recouvrement ouvert de U, tels que $s_{i|V_i\cap V_j} = s_{j|V_i\cap V_j}$ alors il existe $s \in F(U)$ tel que $\forall i\ s_{|V_i} = s_i$.

5.3.2 Spécifications de la propriété d'unicité

On commence par définir les recouvrements ouverts d'une partie A de E, c'est une partie de l'ensemble des ouverts qui doit vérifier la propriété suivante :

Definition.
let covering_prop be
the function defined by , for all P:$\wp$(open of E) ,
A:$\wp$(E) ,
$\forall$ x:E , x $\in$ A $\Rightarrow$ $\exists$ Pi:open of E , (Pi $\in$ P) $\wedge$ (x $\in$ Pi).

Record Open_covering_of [A:$\wp$(E)]: Type := {
Open_covering_setoid: of type $\wp$(open of E) *(coercion)*;
Open_covering_prf: a proof of (covering_prop Open_covering_setoid A) }.

Let U be a open of X.
Let P be of type Open_covering_of(U).
Let s, s' be of type F(U) .

Definition.
let equal_over_a_covering be $\forall$ V:Open_covering_setoid(P) , s|U $\cap$ V = s'|U $\cap$ V.

La propriété ci-dessus est un peu différente de celle donnée dans la définition des faisceaux. En effet on restreint s et s' à $U \cap V$ et pas à V simplement parce que même si V fait partie du recouvrement de U il n'est pas forcément inclus dans U (un recouvrement peut 'déborder'). Cette propriété définie, on peut formaliser la première propriété des faisceaux ainsi :

Definition.
let unicity_presheaf be
the function defined by , for all U:open of X ,
P:Open_covering_of(U) ,
s,s':F(U) , (!equal_over_a_covering U P s s') ⇒ s = s'.

5.3.3 Spécifications de la propriété de recollement

Le problème pour la seconde propriété est de se donner la collection d'éléments s_i. On voudrait considérer le type suivant:

Definition.
a stick_elt is a structure with the following components:
stick_elt_set: a open of X
stick_elt_elt: of type F(stick_elt_set)

en faire un setoïde, puis dire que se donner une telle collection c'est se donner une partie de ce setoïde. On a donc besoin pour cela d'une égalité sur ce type. La notion d'égalité la plus naturelle serait $x = x'$ si `(part x) = (part x')` et `(elt x) = (elt x')`, mais elle ne peut être codée telle quelle parce que les types de `(elt x)`et de `(elt x')`, respectivement `(F(part x))`et `(F(part x'))`, sont différents (`(elt x)`et `(elt x')`n'appartiennent donc pas au même sétoïde).

Remarquons par contre que si `(part x) = (part x')`, on a en particulier l'inclusion `(part x)`$\subset$`(part x')` et `(elt x')` peut donc être vu comme un élément de type `(F (part x))` via les morphismes de restrictions. On dira donc que $x = x'$ si `(part x)=(part x')` et $(elt\ x) = (elt\ x')_{|(partx)}$:

Definition.
let equ_stick_elt be
the function defined by , for all st1,st2:stick_elt ,
∃ proof_equal_sets:st1_SET = st2_SET ,
st1_ELT = st2_ELT|st1_SET.

Lemma (equ_stick_equiv).
equ_stick_elt is an equivalence.
Proof:

Definition.
let Stick_elt be the set Build_Setoid(equ_stick_equiv).

Si `s:Stick_elt` on note sous *CtCoq* s_ELT le terme `(stick_elt_elt s)`et s_SET le terme `(stick_elt_set s)`.

Une fois l'ensemble `Stick_elt` des éléments de la forme (U,$s \in F(U)$) définit, on peut écrire la propriété de recollement. Une donnée de recollement (`sticking_data_on` en *Coq*) sera la donnée d'une partie `st_part` de `Stick_elt` , d'une preuve que l'ensemble des ouverts associés forme bien un recouvrement de U, et d'une preuve que $\forall(U,s_U),(V,s_V) \in st\_part$, $s_{U|U\cap V} = s_{V|U\cap V}$.

Let U be a open of X.
Let P be of type Open_covering_of(U).
Let s, s' be of type F(U) .

Definition.
let equal_over_intersections be
the function defined by , for all Vi,Vj:open of X ,
si:F(Vi) , sj:F(Vj) , si|Vi ∩ Vj = sj|Vi ∩ Vj.

Record sticking_data_on [U:open of X]: Type := {
st_part: of type Predicate(Stick_elt(F)) *(coercion)*;
st_covering_prf: of type ∀ x:U , ∃ st:st_part , x ∈ st_SET;
st_prf: of type ∀ U,V:st_part , (equal_over_intersections U_ELT V_ELT) }.

La propriété de recollement sera alors :

Definition.
let sticking be
∀ U:open of X ,
∀ st:sticking_data_on(U) , ∃ s:F(U) , ∀ Si:st , (equal_over_intersections s Si_ELT).

Et on peut enfin définir les faisceaux d'anneaux commutatifs:

Record CRsheaf [X:topological space]: Type := {
pre: a presheaf of commutative rings over X *(coercion)*;
unicity_prf: of type ∀ U:open of X ,
∀ P:Open_covering_of(U) , ∀ s,s':U , (unicity_presheaf(P) s s');
sticking_prf: of type sticking(pre) }.

5.4 Germes, fibres et morphismes

5.4.1 Germes et fibres

La fibre F_p d'un préfaisceau F en un point p de X est l'ensemble des couples formés d'un ouvert U contenant p et d'un élément s de $F(U)$, quotienté par la relation $(U,s \in F(U)) \sim (V,s' \in F(V))$ ssi $(\exists W |\ p \in W \wedge (W \subset U \cap V) \wedge s_{|W} = s'_{|W})$. La souplesse des setoïdes permet de directement définir les fibres `(Stalk F p)` :

Definition.
a germ is a structure with the following components:
germ_set: a open of X
germ_prf: of type P ∈ germ_set
germ_elt: of type F(germ_set)

Definition.
let equ_germ be
the function defined by , for all g1,g2:germ ,
∃ W:open of X ,
(P ∈ W) ∧
∃ p1:W ⊂ g1$_{_set}$, ∃ p2:W ⊂ g2$_{_set}$, g1$_{_elt|W}$ = g2$_{_elt|W}$.

Lemma (equ_germ_equiv).
equ_germ is an equivalence.
Proof:

Definition.
let stalk be the set Build_Setoid(equ_germ_equiv).

Les anneaux $F(U)$ induisent une structure d'anneau sur les fibres : par exemple on peut définir la somme de deux germes en p $(U,s) + (V,s')$ par $(U \cap V, s_{U\cap V} + s'_{U\cap V})$, il suffit de montrer que cette fonction est compatible avec l'égalité des fibres :

Definition.
stalk_plus:stalk ⇒ stalk ⇒ stalk
1: `Intros g1 g2.`
1: `Apply` (!Build_germ
g1$_{set}$ ∩ g2$_{set}$ (in_part_intero germ_prf(g1) germ_prf(g2))
g1$_{elt|g1_set}$ ∩ g2$_{set}$+g2$_{elt|g1_set}$ ∩ g2$_{set}$).

Lemma (ringstalk1).
∀ x,x',y,y':stalk , x = x' ⇒ y = y' ⇒ (stalk_plus x y) = (stalk_plus x' y').
Proof:

On définit de même la multiplication, le zéro (X,$0 \in F(X)$), l'unité (X,$1 \in F(X)$), puis on montre les propriétés nécessaires pour enfin faire des fibres des anneaux.

5.4.2 Morphismes

Comme pour toutes les structures mathématiques il existe une notion de morphisme pour les préfaisceaux. Un morphisme ϕ entre deux préfaisceaux F et G sur X est la donnée, pour tout ouvert U de X, de morphismes d'anneaux $\phi(U) : F(U) \to G(U)$, compatibles avec les morphismes de restrictions.

$$\begin{array}{rccc} \phi(V): & F(V) & \longrightarrow & G(V) \\ & \downarrow \rho^F_{VU} & \circlearrowleft & \downarrow \rho^G_{VU} \\ \phi(U): & F(U) & \longrightarrow & G(U) \end{array}$$

Definition.
a presheaf_morphism is a structure with the following components:
morphism_of: of type ∀ U:open of X , F(U) ⟶ G(U) *(coercion)*
commut_prf: of type ∀ U,V:open of X , ∀ t:(prf_included V U) , (ρUV o U) = (V o ρUV)

Un tel morphisme ϕ sera dit injectif (resp. surjectif) si, pour tout ouvert U, $\phi(U)$ est injectif (resp. surjectif). Par définition un morphisme de faisceau est un morphisme entre les préfaisceaux sous-jascents : la coercion entre faisceau et préfaisceau permet de ne pas avoir à donner de nouvelle définition.

Enfin, un morphisme de faisceau induit naturellement sur toutes les fibres un morphisme d'anneau $\phi_p : F_p \to G_p$, en envoyant un germe représenté par un couple (U,$s \in (F\ U)$) vers (U,$\phi(U)(s) \in (G\ U)$).

```
Let X be a topological space.
Let F, G be presheaf of commutative rings over X .
Let P be of type X.
Let f be of type (presheaf_morphism F G).
Definition.
    let fun_induced_in_stalks be
     the function defined by , for all germ_f:(Stalk F P) ,
                                       (!Build_germ
                                           X G P germ_f_set germ_prf(germ_f)
                                           f(germ_f_set)(germ_f_elt))
     (its type is (Stalk F P) ⇒ (Stalk G P)).
```

On prouve que cette fonction est bien compatible avec l'égalité sur les germes pour en faire une application entre les fibres, puis toutes les propriétés nécessaires pour en faire un morphisme d'anneau :

```
Lemma (fun_comp_in_stalks).
    fun_compatible(fun_induced_in_stalks).

Lemma (fun_induced_in_stalks_plus).
    ∀ x,y:(Stalk F P) ,
    fun_induced_in_stalksx+y = fun_induced_in_stalks(x)+fun_induced_in_stalks(y).

Lemma (fun_induced_in_stalks_zero).
    fun_induced_in_stalks(0(Stalk F P)) = 0(Stalk G P).

Lemma (fun_induced_in_stalks_mult).
    ∀ x,y:(Stalk F P) ,
    fun_induced_in_stalksx*y = fun_induced_in_stalks(x)*fun_induced_in_stalks(y).

Lemma (fun_induced_in_stalks_one).
    fun_induced_in_stalks(1(Stalk F P)) = 1(Stalk G P).

Definition.
    let map_induced_in_stalks be
     (!BUILD_HOM_RING
         (Stalk F P) (Stalk G P) fun_induced_in_stalks fun_comp_in_stalks
         fun_induced_in_stalks_plus fun_induced_in_stalks_zero fun_induced_in_stalks_mult
         fun_induced_in_stalks_one)
     (its type is (!Hom RING (Stalk F P) (Stalk G P))).
```

Ces morphismes induits `(map_induced_in_stalks f p)` sont ainsi formalisés, on les affiche f_p sous *CtCoq* .

5.5 Nature locale des faisceaux

On a montré le théorème suivant : un morphisme de faisceau est un isomorphisme (au sens de morphisme injectif et surjectif) si tous les morphismes induits dans les fibres sont des isomorphismes (la contraposée est vraie). Ce théorème n'est vrai que pour les faisceaux, il utilise explicitement les deux propriétés qui les définissent. On va détailler une partie de la preuve: le morphisme $\phi : F \to G$ est injectif si et seulement si tous les morphismes induits par ϕ dans les fibres le sont (attention c'est faux pour la surjectivité).

Soit donc U un ouvert et $s \in F(U)$ tel que $\phi(U)(s) = 0$. Si en tout point les morphismes induits sont injectifs alors on va avoir autour de chacun de ces points p un ouvert W_p tel que $s_{|W_p} = 0$:

Lemma (localy_zero).
Phi(U)(s) = 0 ⇒
(∀ P:X , Phi_P is injective) ⇒
∀ P:X ,
∀ t:P ∈ U , ∃ Wp:open of X , ∃ pw:(prf_included Wp U) , (P ∈ Wp) ∧ (s|Wp = 0).

Si on considère l'ensemble $\mathcal{W}$ de ces ouverts W_p, le lemme ci-dessus nous permet de démontrer qu'il forme un recouvrement ouvert de U. Cependant former l'ensemble exacte de ces ouverts ne peut se faire sans utiliser l'axiome du choix (on passe de 'il existe W_p' à 'l'ensemble des W_p'). Pour éviter cela, on construit un ensemble plus gros : celui de tous les ouverts W tels que $s_{|W} = 0$.

Definition.
a_good_setoid: ℘(open of X)
1: `Apply` (!Build_Predicate ? W:open of X , ∃ pu:(prf_included W U) , s|W = 0).

Puisque c'est un sur-ensemble de $\mathcal{W}$, il forme lui aussi un recouvrement de U :

Definition.
a_good_covering:(morphism_of Phi U)(s) = 0 ⇒
(∀ P:X , Phi_P is injective) ⇒ Open_covering_of(U)
1: `Intros` H' H'0. 1: `Apply` (!Build_Open_covering_of ? U)(a_good_setoid).

Par l'application de la première propriété des faisceaux on aura donc $s = 0$, et il vient :

Lemma (injective_phi).
∀ Phi:(presheaf_morphism F G) ,
(∀ P:X , Phi_P is injective) ⇒ ∀ U:open of X , Phi(U) is injective.

5.6 Faisceau associé

On a formalisé de deux façons le faisceau associé à un préfaisceau. La première en suivant la définition de R. Hartshorne [HAR77], la seconde à l'aide de la notion de crible. On décrit ici la version de R. Hartshorne.

5.6.1 La définition de R. Hartshorne

Considérons un préfaisceau F. On définit le faisceau F^* associé à F par les applications suivantes :

- Entre les objets :

  A un ouvert U on fait correspondre l'anneau des applications $s : U \to \bigsqcup_{p \in U} F_p$ qui vérifient les deux conditions suivantes :

  i) $\forall p \in U$, $s(p) \in F_p$

  ii) $\forall p \in U\ \exists V$ *voisinage de* $p\ \ \exists t \in F(V)$ *tel que* $\forall q \in V\ s(q) = (V,t) \in F_q$

- Entre les morphismes :

  Si $U \subset V$ on définit la restriction $F(V) \to F(U)$ par la restriction des applications à U.

R. Hartshorne laisse au lecteur le soin de démontrer que le préfaisceau ainsi défini est en fait un faisceau, ce que l'on s'est proposé de faire en *Coq*.

5.6.2 Des applications dépendantes

L'ensemble des fonctions $s : U \to \bigsqcup_{p \in U} F_p$ qui vérifient $\forall p \in U$, $s(p) \in F_p$ peuvent être vues en *Coq* par les termes de type `(p:(subtype_elt U))(Stalk F (subtype_elt p))`. On va faire de l'ensemble de ces termes un setoïde.

Soit donc f un terme de ce type. Puisque f doit représenter une fonction, on doit avoir : si x=y alors (f x)=(f y), mais là encore cette dernière égalité n'a pas de sens puisque le premier terme est de type `(Stalk F x)`et le second de type `(Stalk F y)`.

De manière générale, on peut procéder de la manière suivante. Soit `E:Setoid` et `F:E ->Setoid`, on demande en paramètre une fonction G qui à x,y:E et une preuve p que x=y associe une fonction `(G x y p):(F x)->(F y)` qui, en quelque sorte, reconstruit un élément de type `(F x)` en un élément de type `(F y)` (on veut évidemment que cette fonction soit indépendante de la preuve de x=y donnée en paramètre):

Let E be a set.
Let F be a function of type E ⇒ set.
Definition.
a map_for_equality is a structure with the following components:
equality_map: of type ∀ x,y:E , ∀ p:x = y , F(x) ⟶ F(y) *(coercion)*
indep_proof_prf: of type ∀ x,y:E ,
∀ pxy,p'xy:x = y ,
(equality_map x y pxy) = (equality_map x y p'xy)

on peut alors définir les applications dont le type d'arrivé dépend ainsi de l'argument par:

Let G be of type map_for_equality(F).
Definition.
a mapd is a structure with the following components:
mapd_fun: of type ∀ x:E , F(x) *(coercion)*
mapd_prf: of type ∀ x,y:E , ∀ prf_equ:x = y , y = (G x y prf_equ)(x)

On met sur ce type l'égalité extentionelle classique:

Definition.
let equ_mapd be the function defined by , for all f1,f2:mapd , ∀ x:E , f1(x) = f2(x)
(its type is mapd ⇒ mapd ⇒ Prop).
Lemma (equ_mapd_equiv).
equ_mapd is an equivalence.

Definition.
let MAPd be the set (!Build_Setoid ? ? equ_mapd_equiv).

Remarque: Cette même idée nous permet aussi d'avoir une égalité significative sur les sigma-types : l'égalité de `(a:A,b:(f a))` et de `(a':A',b':(f a'))` pourra être définie par `∃ p:a=a' | b'=(G a a' p b)`. Pour que cette définition soit valide, c'est-à-dire pour que ce soit une relation d'équivalence, il faut de plus demander à G de vérifier deux propriétés :

- $\forall x : A, \forall p : x = x, (G\ x\ x\ p) = Id_{(fa)}$
- $\forall x,y,z : A, \forall p_{xy} : x = y, \forall p_{yz} : y = z, \forall p_{xz} : x = z, (G\ y\ z\ p_{yz}) o (G\ x\ y\ p_{xy}) = (G\ x\ z\ p_{xz})$

5.6.3 Spécifications et preuves

Dans notre cas, on prends pour E le setoïde `(subtype_elt U)`, et pour F la fonction `point_to_stalk` qui à `p:(subtype_elt U)` associe `(Stalk F (subtype_elt p))`. L'application G sera la fonction `fun_stalk_to_same` qui à partir de `p1,p2:(subtype_elt U)` et d'une preuve p que `p1=p2` reconstruira tout germe de F_{p1} en un germe de F_{p2} :

Definition.
let point_to_stalk be the function defined by , for all P:U , (Stalk F P).
Lemma (in_part_comp_l).
$\forall$ A:$\wp$(E) , $\forall$ x,y:E , x $\in$ A $\Rightarrow$ y = x $\Rightarrow$ y $\in$ A.

Definition.
let fun_stalk_to_same be
the function defined by , for all p1,p2:U ,
prf:p1 = p2 ,
st1:(Stalk F p1) ,
(Build_germ (in_part_comp_l germ_prf(st1) Sym(prf))
st1$_{elt}$)
(its type is $\forall$ p1,p2:U , $\forall$ prf:p1 = p2 , point_to_stalk(p1) $\Rightarrow$ point_to_stalk(p2)).

Pour plus de commodité par la suite on fait de `fun_stalk_to_same` une application qui ne renvoie non pas une simple application de F_{p1} vers F_{p2}, mais un morphisme d'anneau. Après avoir vérifié toute les propriétés nécessaires, on peut enfin définir l'ensemble des fonctions qui nous intéressent par :

Definition.
as_map_equality:(!map_for_equality U point_to_stalk)
1: `Apply` (!Build_map_for_equality U point_to_stalk hom_stalk_to_same).

Definition.
let local_data_map be MAPd(as_map_equality).

Les fonctions qui définissent le faisceau associé doivent en outre vérifier la propriété (ii), on construit donc l'ensemble de ces fonctions comme la partie de `local_data_map` vérifiant :

Definition.
associated_sheaf_setoid:$\wp$(local_data_map)

```
1: Apply (!Build_Predicate ?
          s:local_data_map ,
          ∀ p:U ,
          ∃ V:open of X ,
          ∃ proof_neigh:p ∈ V ,
          ∃ proof_included:V ⊂ U ,
          ∃ t:F(V) , ∀ q:V , s(q[in U]) = (in_stalk F subtype_prf(q))(t)).
```

On fait alors de ce sétoïd un anneau, puis on définit les morphismes de restrictions. On prouve enfin que le préfaisceau ainsi contruit est bien un faisceau.

6 Spectre et Schémas Affines

6.1 Le spectre d'un anneau comme espace topologique

6.1.1 Le spectre d'un anneau comme ensemble

Soit R un anneau commutatif, comme ensemble, son spectre est l'ensemble de ses idéaux premiers (avec bien sûr pour égalité celle des parties de R):

Let R be a commutative ring.
Definition.
spec:set
1: `Apply` (!Build_Setoid prime ideal of R !Equal($\wp$(R))).

6.1.2 La topologie de Zarisky sur (spec R)

Si $\mathfrak{I}$ est un idéal de R, on définit $V(\mathfrak{I})$ (noté de cette manière avec *CtCoq* mais correspondant au terme *Coq* `(primes_containing I)`) comme l'ensemble des idéaux premiers de R qui contiennent $\mathfrak{I}$.

Definition.
primes_containing:$\forall$ I:ideal(R), $\wp$(spec)
1: `Intros` I. 1: `Apply` (!Build_Predicate spec x:spec, I $\subset$ x).

Les ensembles de la forme $V(\mathfrak{I})$ forment en fait les fermés d'une topologie sur (spec R), que l'on va donc construire en utilisant la fonction `Build_topology_by_closed` vue en 3.1. On note $(Spec\ R)$ cet espace topologique.

On commence donc par construire l'ensemble des $V(\mathfrak{I})$:

Definition.
set_of_closed_for_zarisky:$\wp(\wp$(spec))
1: `Apply` (!Build_Predicate ? p:$\wp$(spec), $\exists$ a:Set_Ideal(R), p = V(a)).

On montre facilement que les propriétés `F_empty F_full` de la section 3.1 : le vide s'écrit `(primes_containing (ideal_full R))` (où `(ideal_full R)` est l'anneau R vu comme ideal) et l'ensemble des idéaux premiers de R s'écrit `(primes_containing (ideal_nul R))` :

Lemma (spec_top_prop1).
$\emptyset \in$ set_of_closed_for_zarisky.

Lemma (spec_top_prop2).
full(spec) $\in$ set_of_closed_for_zarisky.

Les propriétés `finite_union_closed` (l'union de deux fermés est un fermé) et `infinite_inter_closed` (une intersection quelconque de fermés est un fermé) reposent essentiellement sur le fait que $i)$ $V(\mathfrak{a}) \cup V(\mathfrak{b}) = V(\mathfrak{a} \cap \mathfrak{b})$ et $ii)$ $\cap V(\mathfrak{a}_i) = V(\sum \mathfrak{a}_i)$, ce qu'il nous faut donc démontrer.

La première égalité $i)$ est une conséquence du lemme `inter_included_prime` vu dans la section `4.1` : si un idéal premier contient l'intersection de deux idéaux, alors il contient l'un ou l'autre de ces idéaux.

Lemma (union_primes_containing).
∀ I,J:ideal(R) , V(I) ∪ V(J)=V(I∩J).

Lemma (spec_top_prop4).
finite_union_closed(set_of_closed_for_zarisky).

Pour exprimer la propriété $ii)$, on définit d'abord la fonction qui associe à une famille d'idéaux $\{\mathfrak{a}_i\}$ l'ensemble des $V(\mathfrak{a}_i)$:

Definition.
family_of_primes_containing:Predicate(Set_Ideal(R)) ⇒ Predicate(℘(spec))
1: `Intros` Ai. 1: `Apply` (!Build_Predicate ℘(spec) p:Predicate(spec) , ∃ ai:Ai , p = V(ai)).

On peut alors démontrer, en utilisant les lemmes qui caractérisent la somme d'idéaux ($\sum \mathfrak{a}_i$ est le plus petit idéal qui contient tous les $\mathfrak{a}_i$) :

Lemma (primes_containing_sum_ideals).
∀ ideal_collection: ℘(Set_Ideal(R)) ,
V(sum_ideals(ideal_collection)) =
intersection_part(family_of_primes_containing(ideal_collection)).

On a donc bien :

Lemma (spec_top_prop3).
infinite_inter_closed(set_of_closed_for_zarisky).

Et on peut définir le spectre d'un anneau:

Definition.
let spec_top be
(!Build_topology_by_closed
spec set_of_closed_for_zarisky spec_top_prop1 spec_top_prop2 spec_top_prop3
spec_top_prop4).
Definition.
let Spec be the topological space Build_et(spec_top).

6.2 Un préfaisceau particulier pour définir les schémas affines

Le schéma affine défini par R est l'espace topologique $(Spec\ R)$ muni du faisceau $\mathcal{O}_{(SpecR)}$, qui sera le faisceau associé au préfaisceau $\mathfrak{F}$ que l'on décrit ici.

Pour tout ouvert U de $(Spec\ R)$ on définit $\mathfrak{F}(U)$ comme l'anneau constitué des couples (g,h) de R^2 tels que $\forall p \in U, h \notin p$. L'égalité, la somme et le produit sont définit comme pour les fractions, par exemple $(g_1,h_1)+(g_2,h_2) = (g_1 * h_2 + g_2 * h_1, h_1 * h_2)$, le zéro est $(0_R,1_R)$, et l'unité $(1_R,1_R)$.

```
Let U be a open of Spec(A).
Definition.
a spec_pre_elt is a structure with the following components:
sp_num: of type A
sp_den: of type A
sp_prf: of type ∀ p:prime ideal of A , p ∈ U ⇒ sp_den ∉ p


Definition.
    let equ_spec_pre be
     the function defined by , for all s,s':spec_pre_elt , snum*s'den = s'num*sden
     (its type is spec_pre_elt ⇒ spec_pre_elt ⇒ Prop).
Definition.
    spec_pre_set:set
 1: Apply (!Build_Setoid spec_pre_elt equ_spec_pre).
Definition.
    spec_pre_plus:spec_pre_elt ⇒ spec_pre_elt ⇒ spec_pre_elt
 1: Intros s s'. 1: Apply (!Build_spec_pre_elt snum*s'den+s'num*sden sden*s'den).
Definition.
    spec_pre_zero:spec_pre_elt
 1: Apply (!Build_spec_pre_elt 0 1).
```

Les restrictions sont définies de manière naturelle : si $U \subset V$ on envoi un couple (g,h) de $\mathfrak{F}(V)$ sur ce même couple vu comme élément de $\mathfrak{F}(U)$, la condition $\forall p \in U, h \notin p$ étant naturellement satisfaite puisque $U \subset V$.

```
Definition.
    spec_pre_morph_fun:∀ prf_incl:(!Hom Top_cat(Spec(A)) U V) ,
                          spec_pre_cring(V) ⇒ spec_pre_cring(U)
 1: Intros prf_incl s. 1: Apply (!Build_spec_pre_elt U snum sden).
```

On en fait une application (en prouvant la compatibilité), puis un morphisme d'anneau :

Definition.
spec_pre_morph:∀ prf_incl:(!Hom Top_cat(Spec(A)) U V) ,
(!Hom CRING spec_pre_cring(V) spec_pre_cring(U))
1: `Intros` prf_incl. 1: `Apply` (!Build_ring_hom ? ? spec_pre_morph_map(prf_incl)).

Definition.
spec_pre_morph_map_map:∀ U,V:Ob(Top_cat(Spec(A))) ,
(!Hom Top_cat(Spec(A)) U V) ⇝
(!Hom CRING spec_pre_cring(V) spec_pre_cring(U))
1: `Intros` U V. 1: `Apply` (!Build_Map ? ? (!spec_pre_morph U V)).

On peut alors enfin construire, en prouvant les deux propriétés des foncteurs, le préfaisceau $\mathfrak{F}$:

Definition.
spec_presheaf:presheaf of commutative rings over Spec(A)

1: `Apply` (!Build_Cfunctor
Top_cat(Spec(A)) CRING spec_pre_cring spec_pre_morph_map_map).

Une fois le préfaisceau $\mathfrak{F}$ (`(spec_presheaf R)` en *Coq*) formalisé, on peut enfin définir le faisceau $\mathcal{O}_{(SpecR)}$ par `(associated_sheaf (spec_presheaf R))`.

7 Problèmes rencontrés

Deux principaux problèmes ont considérablement ralentit l'évolution de ce dévellopement. Ils imposent de plus une certaine différence avec la 'manière de faire' en mathématique.

7.1 Limitations des coercions

Le premier concerne les limitations des coercions autorisées par *Coq*. En effet les coercions paramétrées ne sont pas gérées par le système et cela oblige l'utilisateur à explicitement faire appel à la fonction `subtype_elt` pour voir un élément d'une partie comme élément de l'ensemble au-dessus. Ce qui peut sembler anecdotique devient vite un véritable puzzle avec l'empilement des différentes structures et théories dans *Coq*. Si on considère par exemple le terme servant à définir une des propriétés du faisceau associé, affiché ainsi par *Ctcoq* :

1: `Apply` (!Build_Predicate ?
asso_fun:local_data_map ,
∀ P:U ,
∃ V:open of X ,
∃ proof_neigh:P ∈ V ,
∃ proof_included:V ⊂ U ,
∃ t:F(V) , ∀ Q:V , asso_fun(Q[in ?]) = (in_stalk F subtype_prf(Q))(t)).

il correspond en fait au code *Coq* suivant :

```
Apply (!Build_Predicate ?
         [asso_fun:local_data_map] (P:(subtype_elt U))
         (exT   (open X) [V:(open X)]
            (exT  (in_part (subtype_elt P) (subtype_elt V))
               [proof_neigh:(in_part (subtype_elt P) (subtype_elt V))]
               (exT (included (subtype_elt V) (subtype_elt U))
                  [proof_included:(included (subtype_elt V) (subtype_elt U))]
                  (exT (F V) [t:(F V)]
                     (Q:(subtype_elt V))
                      (Equal
                        (asso_fun
                           (!Build_subtype ? ? (subtype_elt Q)
                            (proof_included (subtype_elt Q) (subtype_prf Q))))
                        ((in_stalk F (subtype_prf Q)) t)))))).
```

et contient pas moins de douze fois la fonction `subtype_elt`! On remarque de plus que les possibilités étendues de notations de *Ctcoq* sont essentielles pour comprendre les termes que l'on manipule.

7.2 Les setoïdes

La seconde difficulté vient elle du choix même de la structure utilisée pour représenter les ensembles : les sétoïdes. S'ils sont pour le moment nécessaires pour permettre le travail avec des ensembles quotients, ils obligent à travailler avec une égalité au-dessus de celle de *Coq*. Les étapes de réécriture dans les preuves, très courantes, se font donc par application de lemmes de compatibilité successifs, ce qui est très coûteux en terme de mémoire et d'efficacité.

Ce problème, ajouté à l'empilement des différentes structures mathématiques et des différentes coercions, ralentit considérablement *Coq*. Il n'a par exemple pas été rare d'avoir à attendre de cinq à quinze minutes l'exécution d'une tactique (étape élémentaire de preuve) dans les démonstrations relatives au faisceau associé.

De plus, utiliser les sétoïdes oblige à démontrer les lemmes de compatibilité pour toute nouvelle application, même (et surtout, parce que le cas est bien plus courant) lorsque les sétoïdes ne représentent pas de structure quotient. La encore, ce n'est pas que prouver ces lemmes soit difficile, mais que leur nombre (chaque proposition, chaque fonction doit être compatible) et leur inutilité (lorsque l'ensemble que l'on considère n'est pas un ensemble quotient, il ne devrait pas être nécessaire de les démontrer) rend le travail de formalisation extrèmement fastidieux.

8 Conclusion

Le développement présenté dans cet article représente 19000 lignes de codes, et permet donc de formaliser une des toutes premières notions de la géométrie algébrique, les schémas affines.

Le mécanisme de coercions, ainsi que la synthèse automatique des arguments implicites dans un terme se sont avérés être des atouts primordiaux tout au long de ce travail, allégeant à la fois la lecture des énoncés, l'écriture des termes, et la compréhension de ceux-ci par l'utilisateur.

La taille et la complexité des termes considérés grandissent de manière non négligeable tout au long de l'avancée de notre travail (jusqu'à plus d'une dizaine de lignes de code pour certains termes). En cela les spécificités de *CtCoq* (possibilités de notations étendues mais surtout d'édition structurée du code), ont été nécessaires pour mener à bien notre projet.

Le manque de soustypes (qui permettraient de ne plus utiliser la fonction `subtype_elt`) et de types quotients (qui nous permettraient de ne plus utiliser les setoïdes) sont les principaux problèmes qui ont ralentit notre travail.

Les schémas affines ne forment évidemment qu'une infime partie du corpus de la géométrie algébrique, mais la définition de la notion générale de schémas et de leurs premières propriétés géométriques ne parait pas inaccessible. La difficulté résidera sans doute dans les notions de recollement et d'isomorphisme qui devront être formalisés de manière assez souple pour être réellement utilisables.

Références

[BKT94] Yves Bertot, Gilles Kahn, and Laurent Théry, "Proof by Pointing", Symposium on Theoretical Aspects Computer Science (STACS), Sendai (Japan), LNCS 789, Avril 1994.

[COQ99] L'équipe *Coq*. The Coq Proof Assistant Reference Manual v6.3. `http://pauillac.inria.fr/coq/doc-fra.html`

[CRO96] Projet CROAP, INRIA Sophia-Antipolis. The Ctcoq system . `http://www-sop.inria.fr/croap/ctcoq/ctcoq-fra.html`

[GOD73] Roger Godement, Topologie Algébrique et théorie des faisceaux, Editions Hermann, 1973.

[HAR77] R. Hartshorne, Algebraic Geometry, Springer Verlag, 1977.

[LAN84] Serge Lang, Algebra (second edition), Addison-Wesley Publishing Company, 1984.

[POT99] Loïc Pottier. Basic notions of algebra, Disponible à l'adresse : `http://pauillac.inria.fr/coq/contribs/algebra.html`

[SAI95] A. Saibi and G. Huet, "Constructive Category Theory" In Proceedings of the joint CLICS-TYPES Workshop on Categories and Type Theory, Goteborg (Sweden), January 95.

Table des matières

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399